Real-Time Systems

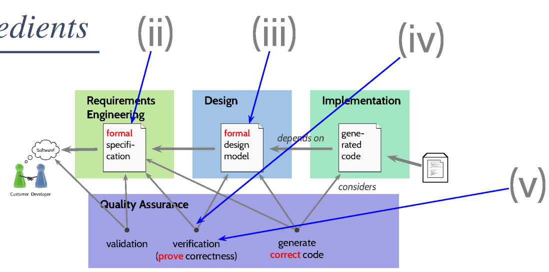
Lecture 2: Timed Behaviour

2017-10-19

Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Necessary Ingredients



To develop **software that is (provably) correct wrt. its requirements**, we need:

(i) a **formal model** of software **behaviour**

(ii) a **language*** to specifiy **requirements** on **behaviour**,
(to distinguish desired from undesired behaviour),

(iii) a **language*** to specify **behaviour** of **design ideas**,

(iv) a notion of **correctness**
(a relation between requirements and design specifications),

(v) and a **method** to **verify (or prove) correctness**
(that a given pair of requirements and design specifications are in correctness relation).

$*$: at best concisely and conveniently, with adequate expressive power.

Content

- **A formal model of real-time** behaviour
  - **state variables** (or **observables**)
  - **evolution** over time (or **behaviour**)
  - **discrete time** vs.
    continous (or **dense**) **time**
- **Timing diagrams**
- **Formalising requirements**
  - with available tools:
    logic and analysis
  - **concise? convenient?**
- **Correctness** of designs wrt. requirements
- **Classes of timed properties**
  - **safety** and **liveness** properties
  - **bounded response** and **duration** properties
- **An outlook to Duration Calculus**

A Formal Model of Real-Time Behaviour

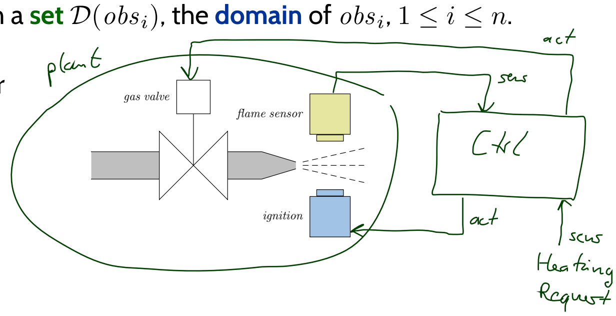
State Variables (or Observables)

- We assume that the real-time systems we consider are characterised by a finite (!) set of **state variables** (or **observables**)

$$obs_1, \ldots, obs_n$$

each associated with a **set** $\mathcal{D}(obs_i)$, the **domain** of obs_i, $1 \le i \le n$.

- **Example**: gas burner



- G

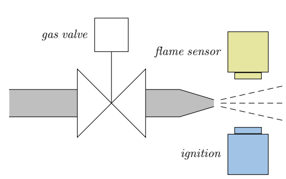
State Variables (or Observables)

- We assume that the real-time systems we consider are characterised by a finite (!) set of **state variables** (or **observables**)

$$obs_1, \ldots, obs_n$$

each associated with a **set** $\mathcal{D}(obs_i)$, the **domain** of obs_i, $1 \le i \le n$.

- **Example**: gas burner



- G , $\mathcal{D}(G) = \{0,1\}$ – domain value 0 for G models "valve closed" (value 1: "valve open")
  (**shorthand notation**: $G : \{0,1\}$)
- $F : \{0,1\}$ – domain value 0 models "no flame sensed" (value 1: "flame sensed")
- $I : \{0,1\}$ – domain value 0 models "ignition device disabled" (value 1: "ignition enabled")
- $H : \{0,1\}$ – domain value 0 models "no heating request sensed" (value 1: "heating request")

Levels of Detail

We can describe a real-time system at various **levels of detail** by **choosing** an **appropriate domain** for each observable.

For example,

- if we need to model a gas valve with **different positions** (not only "open" and "closed"), we could use

  $$G : \{0, 1, 2\} \quad (0\text{: "fully closed"}, 1\text{: "half-open"}, 2\text{: "fully open"})$$

  (Note: domains are never continuous in the lecture, otherwise it's a hybrid system!)
- if the thermostat (sending heating requests) and the gas burner controller are connected via a bus and **exchange messages** from Msg, use

  $$B : Msg^*$$

  to model gas burner controller's receive buffer as a finite sequence of messages from Msg.
- etc.

- Choice of observables and their domain is **a creative (modelling) act**.

  A choice is **good** if it conveniently serves the **modelling purpose**.

System Evolution over Time

- **One** possible **evolution** (over time), or: **behaviour**, of the considered real-time system is represented as a function

  $$\pi : \mathsf{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n).$$

  where Time is the **time domain** ($\to$ in a minute).

- If (and only if) observable obs_i has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in \mathsf{Time}$, $1 \leq i \leq n$, we set

  $$\pi(t) = (d_1, \dots, d_n).$$

- For convenience, we use

  $$obs_i : \mathsf{Time} \to \mathcal{D}(obs_i)$$

  to denote the projection of π onto the i-th component.

What's the time?

- There are two main choices for the time domain Time:
  - **discrete time**: Time $= \mathbb{N}_0$, the set of natural numbers.
  - **continuous or dense time**: Time $= \mathbb{R}_0^+$, the set of non-negative real numbers.

- Throughout the lecture we shall use the **continuous** time model and consider **discrete** time as a special case.

  Because
  - plant models usually live in **continuous** time,
  - we avoid too early introduction introduction of hardware considerations,

- Interesting view: continous-time is a well-suited **abstraction** from the discrete-time realms induced by clock-cycles etc.

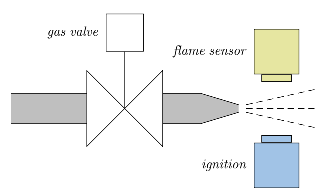
Example: Gas Burner

An evolution over time of the considered real-time system is represented as function

$$\pi : \mathsf{Time} \to \mathcal{D}(obs_1) \times \dots \times \mathcal{D}(obs_n)$$

with $\pi(t) = (d_1, \dots, d_n)$ if (and only if) observable obs_i has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in \mathsf{Time}$, $1 \leq i \leq n$.

For convenience: use $obs_i : \mathsf{Time} \to \mathcal{D}(obs_i)$.



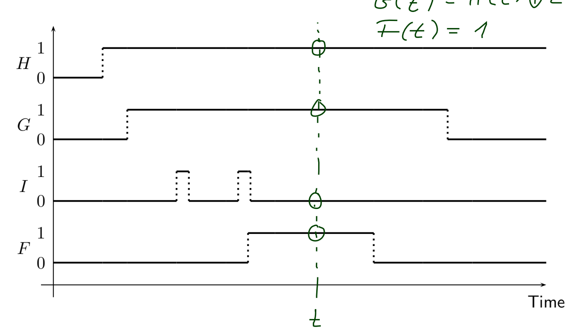
$obs_1 = H$ $obs_2 = G$ $obs_3 = I$ $obs_4 = F$

$\pi(t) = (1, 1, 0, 1)$

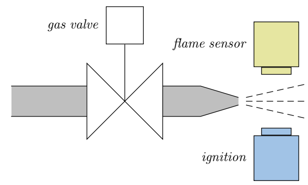
$G(t) = \pi(t) \downarrow 2 = 1$

$F(t) = 1$

π :



Example: Gas Burner

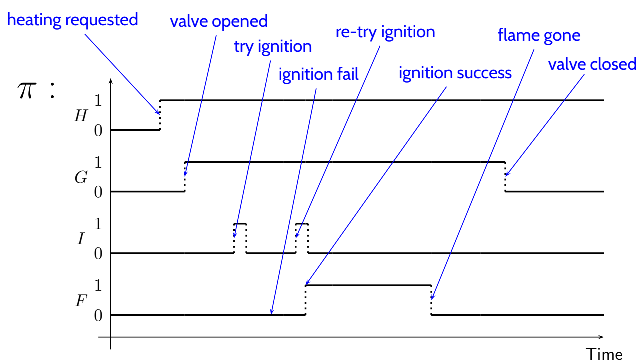


An evolution over time of the considered real-time system is represented as function

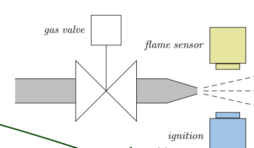
$$\pi : \text{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n)$$

with $\pi(t) = (d_1, \ldots, d_n)$ if (and only if) observable obs_i has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in$ Time, $1 \leq i \leq n$.

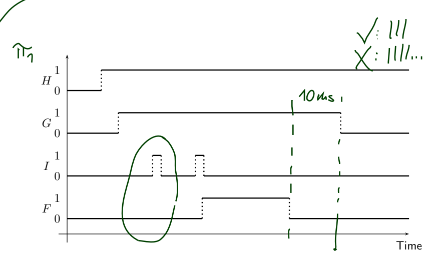
For convenience: use $obs_i : \text{Time} \to \mathcal{D}(obs_i)$.



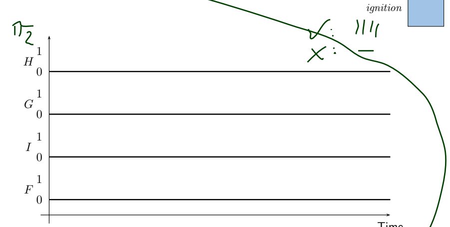
More Examples: Gas Burner Evolutions



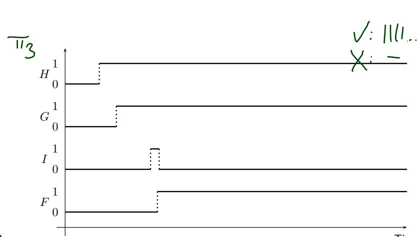
Reg



One ignition failure, success, flame failure.



No heating request, no heating.



Reliable ignition, stable flame.

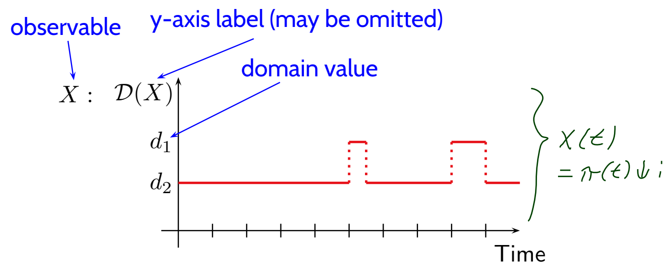
Spontaneous flame, without request.

$\overline{Reg}$

Representing Evolutions: Timing Diagram

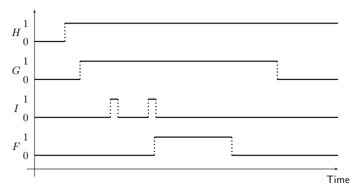
- An **evolution** (of a state variable) can be displayed in form of a **timing diagram**.

- For instance,



for $X : \{d_1, d_2\}$.

- Multiple observables can be combined into a single timing diagram:



Content

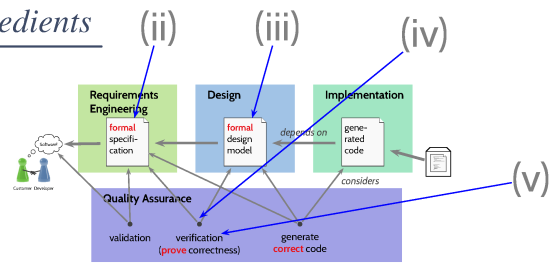
- **A formal model of real-time** behaviour
  - **state variables** (or **observables**)
  - **evolution** over time (or **behaviour**
  - **discrete time** vs.
    continous (or **dense**) **time**

- **Timing diagrams**
- **Formalising requirements**
  - with available tools:
    logic and analysis
  - **concise? convenient?**

- **Correctness** of designs wrt. requirements
- **Classes of timed properties**
  - **safety** and **liveness** properties
  - **bounded response** and **duration** properties

- **An outlook to Duration Calculus**

Necessary Ingredients



To develop **software that is (provably) correct wrt. its requirements**, we need:

(i) a **formal model** of software **behaviour** ✓

(ii) a **language*** to specifiy **requirements** on **behaviour**,
(to distinguish desired from undesired behaviour),

(iii) a **language*** to specify **behaviour** of **design ideas**,

(iv) a notion of **correctness**
(a relation between requirements and design specifications),

(v) and a **method** to **verify (or prove) correctness**
(that a given pair of requirements and design specifications are in correctness relation).

$*$: at best concisely and conveniently, with adequate expressive power.

Formalising Requirements: A First Approach with Available Tools

Requirements, More Formally

- A **requirement** 'Req' is a set of system behaviours (over observables) with the pragmatics that,
  - a **design** or **implementation** is correct wrt. 'Req'
  - if and only if all observed behaviours (of the design or impl.)
  - lie within the set 'Req'.
- More formally,
  - Req $\subseteq (\mathsf{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n))$

    ('Req' is the set of allowed evolutions),
  - let

    $$\mathsf{Des} \subseteq (\mathsf{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n))$$

    be the behaviours of a **design** or **implementation**;
  - 'Des' is **correct** wrt. 'Req' **if and only if** Des $\subseteq$ Req.
- **Inconvenient**:
  'Req' is usually an **infinite** set – we need ways to describe 'Req' conveniently.

Available Tools: Logic and Analysis

- A **requirement** on **gas burner controller** behaviours could be

  "do not ignite if the valve is closed".
- Thus, a **design** 'Des' is correct if
  - for all evolutions $\pi \in$ Des,
    - for all points in time $t \in$ Time,
      - it is not the case that $I(t) = 1$ and $G(t) = 0$.
        (Recall: $I(t)$ is the projection of $\pi(t)$ on the I-component.)
- We can already formalise the above requirement using a **logical formula**:

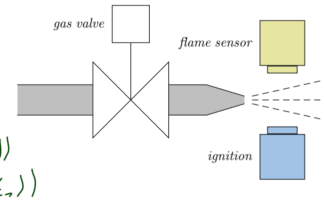
  like : or .

  $$F := \forall\, t \in \mathsf{Time} \bullet \neg(I(t) = 1 \wedge G(t) = 0).$$
- Then Req $= \{\pi : \mathsf{Time} \to \mathcal{D}(H) \times \mathcal{D}(G) \times \mathcal{D}(I) \times \mathcal{D}(F) \mid \pi \models F\}$.
- In the following, we may **identify** a **requirement** and a **logical formulae** which defines the requirement. We say "requirement F".
  IAW: predicate logic formula F serves as **concise description** of requirement 'Req'.

Example: Gas Burner

Req $:\iff \forall\, t \in \mathsf{Time} \bullet \neg(I(t) \wedge \neg G(t))$

$\pi \in$ Req?

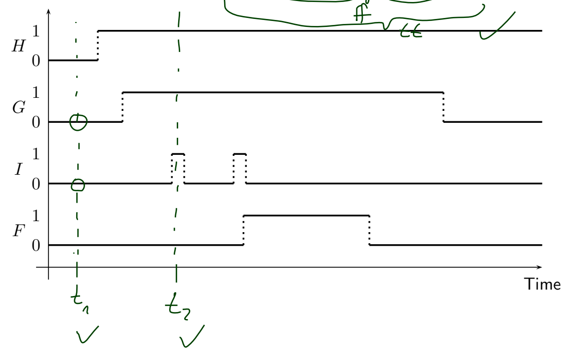


$\pi, \{t \mapsto t_1\} \models \neg(I(t) \wedge \neg G(t))$

$\pi \models \neg(\underbrace{\underbrace{I(t_1)}_{=0}}_{ff} \wedge \underbrace{\neg \underbrace{G(t_1)}_{=0}}_{tt})$

$\underbrace{\qquad\qquad}_{ff}$ tt ✓

π :



Correctness

- Let 'Req' be a **requirement**,
- 'Des' be a **design**, and
- 'Impl' be an **implementation**.

Recall: each is a set of evolutions, i.e. a subset of $(\mathsf{Time} \to \times_{i=1}^{n} \mathcal{D}(obs_i))$.

We say

- 'Des' is a **correct design** (wrt. 'Req') if and only if

$$\mathsf{Des} \subseteq \mathsf{Req}.$$

- 'Impl' is a **correct implementation** (wrt. 'Des' (or 'Req')) if and only if

$$\mathsf{Impl} \subseteq \mathsf{Des} \quad (\text{or } \mathsf{Impl} \subseteq \mathsf{Req})$$

If 'Req' and 'Des' are described by formulae of first-oder predicate logic, proving the design correct amounts to proving validity of

$$\models \mathsf{Des} \implies \mathsf{Req}.$$

Content

- **A formal model of real-time** behaviour
  - **state variables** (or **observables**)
  - **evolution** over time (or **behaviour**
  - **discrete time** vs.
    continous (or **dense**) **time**
- **Timing diagrams**
- **Formalising requirements**
  - with available tools:
    logic and analysis ✓
  - **concise? convenient?**
- **Correctness** of designs wrt. requirements
- **Classes of timed properties**
  - **safety** and **liveness** properties
  - **bounded response** and **duration** properties
- **An outlook to Duration Calculus**

Classes of Timed Properties

Safety Properties

- A **safety property** states that

  something bad must never happen [Lamport].

- **Example**: "do not ignite if the valve is closed"

$$\mathsf{Req} := \forall\, t \in \mathsf{Time} \bullet \neg(I(t) \wedge \neg G(t)).$$

  is a **safety property**.

- In general, a safety property is characterised as a property that can be **falsified** in bounded time:
  - If a gas burner controller does not satisfy 'Req', there is an evolution π and a time $t \in \mathsf{Time}$ such that

$$\neg(I(t) \wedge \neg G(t))$$

    does not hold. All later times $t' > t$ do not make it better.

- But safety is not everything...

Liveness Properties

- The simplest form of a **liveness property** states that

  something good eventually does happen.

- **Example**: "heating requests are finally served"

$$\forall\, t \in \mathsf{Time} \bullet (\underbrace{H(t)}_{H(t)=1} \wedge \neg F(t)) \implies (\exists\, t' \geq t \bullet G(t') \wedge I(t'))$$

  is a **liveness property**.

Note: a gas burner controller can guarantee that finally the valve is opened and ignition is enabled – but **a flame cannot be guaranteed**.

- **Note**: liveness properties not falsified in finite time.
  - if there is a heating request at time t, and at time $t' > t$, the controller did not enforce $G(t) \wedge I(t)$, there may be a later time $t'' > t'$ where the formula holds.

- With real-time systems, liveness is too weak...

Bounded Response Properties

- A **bounded response property** states that

  the desired reaction on an input occurs in time interval $[\overset{\text{begin}}{b}, \overset{\text{end}}{e}]$.

- **Example**: heating requests are served within 3 seconds $\pm\varepsilon$

  $$\forall\, t \in \mathsf{Time} \bullet (H(t) \land \neg F(t)) \implies (\exists\, t' \in [\underbrace{t + 3\,s - \varepsilon}_{b}, \underbrace{t + 3\,s + \varepsilon}_{e}] \bullet G(t') \land I(t'))$$

  is a **bounded liveness property**.

  Here, the interval is $[b, e] = [t + 3\,s - \varepsilon, t + 3\,s + \varepsilon]$;
  it depends on the time t of the heating request.

- This property can again be falsified in finite time.

- With gas burners, this is still not everything...

By the Way: Convenience

It is **not so easy** to read out

"Heating requests are served within 3 seconds $\pm\varepsilon$."

from (lengthy) formula

$$\forall\, t \in \mathsf{Time} \bullet (H(t) \land \neg F(t)) \implies (\exists\, t' \in [t + 3\,s - \varepsilon, t + 3\,s + \varepsilon] \bullet G(t') \land I(t')).$$

The **Duration Calculus** formula

$$((\lceil H \land \neg F \rceil \,;\, \mathit{true}) \land \lceil \neg(G \land I) \rceil) \implies 3 - \varepsilon \le \ell \le 3 + \varepsilon$$

is **more concise** (fewer symbols),
and considered **easier to read out** by some.

$\rightarrow$ in a week.

Duration Properties

- A **duration property** states that
  - for observation interval $[b,e]$ characterised by a condition $A(b,e)$,
    - the **accumulated time**
    - in which the system is in a certain critical state characterised by condition $C(t)$
    - has an upper bound $u(b,e)$.

Riemann integral

$$\forall\, b,e \in \mathsf{Time} \bullet A(\overset{b,e}{\cancel{e,b}}) \implies \left(\int_b^e C(t)\,dt\right) \le u(b,e)$$

- **Example**: leakage in gas burner,

  "At most 5% of any at least 60s long interval amounts to leakage."

$$\forall\, b,e \in \mathsf{Time} \bullet \underbrace{(b \le e \wedge (e-b) \ge 60)}_{A(b,e)} \implies \left(\int_b^e \underbrace{G(t) \wedge \neg F(t)}_{C(t)}\,dt\right) \le \underbrace{\left(0.05 \cdot (e-b)\right)}_{u(b,e)}$$

  is a **duration property**.

Duration Properties

- A **duration property** states that
  - for observation interval $[b,e]$ characterised by a condition $A(b,e)$,
    - the **accumulated time**
    - in which the system is in a certain critical state characterised by condition $C(t)$
    - has an upper bound $u(b,e)$.

Riemann integral

$$\forall\, b,e \in \mathsf{Time} \bullet A(\overset{b,e}{\cancel{e,b}}) \implies \int_b^e C(t)\,dt \le u(b,e)$$

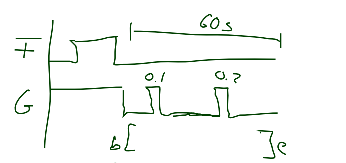
- **Example**: leakage in gas burner,

  "At most 5% of any at least 60s long interval amounts to leakage."

$$\forall\, b,e \in \mathsf{Time} \bullet (b \le e \wedge (e-b) \ge 60) \implies \int_b^e G(t) \wedge \neg F(t)\,dt \le 0.05 \cdot (e-b)$$

  is a **duration property**.

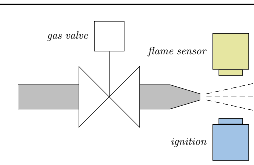
- This property can again be falsified in finite time.



$$\int_b^e G(t) \wedge \neg F(t)\,dt = 0.3$$

An Outlook to Duration Calculus (DC)

Duration Calculus: Preview

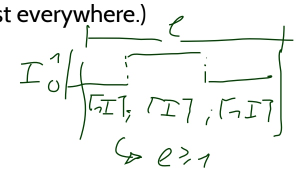


- $G, F, I, H : \{0, 1\}$
- Define $L : \{0, 1\}$ as $G \wedge \neg F$.

- Duration Calculus is an **interval logic**.
- Formulae are evaluated in an (**implicitly given**) interval.

Strangest operators: ⌈Form⌉

- **almost everywhere** – Example: $\lceil G \rceil$
  (Holds in a given interval $[b, e]$ iff the gas valve is open almost everywhere.)
- **chop** ; – Example: $(\lceil \neg I \rceil \, ; \lceil I \rceil \, ; \lceil \neg I \rceil) \implies \ell \geq 1$
  (Ignition phases last at least one time unit.)
- **integral** $\int$ – Example: $\ell \geq 60 \implies \int L \leq \frac{\ell}{20}$
  (At most 5% leakage time within intervals of at least 60 time units.)



Content

- **A formal model of real-time** behaviour
  - **state variables** (or **observables**)
  - **evolution** over time (or **behaviour**
  - **discrete time** vs. **continous** (or **dense**) **time**
- **Timing diagrams**
- **Formalising requirements**
  - with available tools: **logic and analysis**
  - **concise? convenient?**
- **Correctness** of designs wrt. requirements
- **Classes of timed properties**
  - **safety** and **liveness** properties
  - **bounded response** and **duration** properties
- **An outlook to Duration Calculus**

Tell Them What You've Told Them...

- **Evolutions** over **state variables**
  - are a (simple but powerful) **formal model** of timed behaviour, and
  - can be represented by **timing diagrams**.
- A **requirements specification** denotes a set of **desired** behaviours.
- Example **classes** of properties are
  - **safety**: something bad never happens,
  - **liveness**: something good finally happens,
  - **bounded response**: good things happen with deadlines,
  - **duration**: critical conditions have limited duration.
- Real-time requirements **can be formalised** using just **logic and analysis**.

  But: these specifications easily become **hard to read**.
- Something **more concise** and **more readable** (?): **Duration Calculus** ($\rightarrow$ next week)

References

References

Olderog, E.-R. and Dierks, H. (2008). *Real-Time Systems - Formal Specification and Automatic Verification*. Cambridge University Press.